WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Monday, March 06, 2006

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ	
☐	L13	6865607.pn.	2
☐	L12	proxy near8 (intercept or intercepting) near8 (method call) near8 application	2
☐	L11	proxy near8 (intercept or intercepting) near8 (method call)	15
☐	L10	L9 or l8	22
☐	L9	L8 and @AD<20010628	19
☐	L8	L7 and l1	22
☐	L7	(override or overridden) near3 method	1729
☐	L6	L5 and @AD<20010628	21
☐	L5	L4 and l1	26
☐	L4	(call (intercept or intercepting or method))	5823
☐	L3	((base class) or base-class) near4 (proxy or object) near4 interface same (call (intercept or intercepting or method))	1
☐	L2	((base class) or base-class) near4 (proxy or object) near4 interface near8 (call (intercept or intercepting or method))	1
☐	L1	((base class) or base-class) near4 (proxy or object) near4 interface	110

END OF SEARCH HISTORY

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L10: Entry 8 of 22 File: USPT Jan 30, 2001

DOCUMENT-IDENTIFIER: US 6182155 B1
TITLE: Uniform access to and interchange between objects employing a plurality of access methods

Abstract Text (1):
Uniform access to and interchange between objects with use in any environment that supports interface composition through interface inheritance and implementation inheritance from a common base class is provided. Proxies are used to provide both cross-language and remote access to objects. The proxies and the local implementations for objects share a common set of interface base classes, so that the interface of a proxy for an object is indistinguishable from a similar interface of the actual implementation. Each proxy is taught how to deal with call paramters that are proxies of the other kind. A roster of language identifiers is developed, and a method is added to each object implementation which, when called, checks whether it matches the language that the object implementation is written in. If so, it returns a direct pointer to the object implementation. Common client coding can then be used to deal with both same language and cross-language calls.

Application Filing Date (1):
19980102

Detailed Description Text (8):
FIG. 2B illustrates how an implementation 28 may equally be the target of a reference. The implementation inherits from a C++ proxy class and overrides the methods provided in the proxy classes. When client 30 calls a method on this composite object, the C++ virtual function mechanism ensures that the override methods provided by the implementation are invoked in preference to the methods supplied by the proxy classes. The Skeleton class 32 helps provide the Dispatcher with capability for an ORB to use and is not relevant in the case where the client is local to the implementation.

Detailed Description Text (11):
Parallel hierarchies of generated proxy classes 48 and 52 inherit from the pure interface class hierarchy and provide, by means of C++ virtual function overrides, implementations of the methods appropriate to different kinds of proxies. By way of example, class hierarchy 48 provides ORB proxy method implementations, and common ORB proxy base class 50 introduces instance data and common method implementations necessary for the proper operation of the ORB with this proxy object but not necessary for the proper operation of other proxy kinds. Class hierarchy 52 provides local cross-language proxy method implementations that make use of the run time facilities described in the above referenced application, "Transparent Use of Compiled or Interpreted Objects in an Object Oriented System", and its common base class 54 introduces instance data necessary for the proper operation of a local cross-language proxy but not necessary for the operation of other proxy kinds.

Detailed Description Text (29):
FIG. 5 schematically illustrates the C++ class inheritance structure pursuant to the present invention which satisfies these requirements. A set of interface classes 180 is generated from IDL describing the interfaces desired by the user.

These generated interface classes inherit from the common interface base class 182, which in the preferred embodiment is the CORBA::Object class. In this class are introduced instance data and methods that are common to all proxies and implementations. In particular, the instance datum "m_somref" 184 is introduced here, as are the methods "_SOMProxy ( )" 186 and "_ORBProxy( )" 188 which return pointer values. Default implementations of the methods are supplied that return null pointers.

Detailed Description Text (30):
A set of ORB proxy classes 190 is also generated, each proxy class of which inherits from a corresponding member of the set of interface classes. The generated ORB proxy classes also inherit from a common ORB proxy base class 192, which in the preferred embodiment is the CORBA::Object_ORBProxy class. In the common ORB proxy base class are introduced the instance data and methods that are common to ORB proxies and ORB-accessible implementations, the inclusion of which renders an object ORB enabled. The ORB proxy base class also overrides the " ORBProxy( )" method and causes that method to return a pointer to the ORB proxy base class, instead of a null pointer.

Detailed Description Text (32):
A Skeleton class 200 is also generated, that inherits from the interface class hierarchy and from the ORB proxy base class. A C++ Implementation class 202, constructed by the user, inherits from this class.

Detailed Description Text (36):
A set of SOM proxy classes 210 is also generated, each proxy class of which inherits from a corresponding member of the set of interface classes 180. The generated SOM proxy classes also inherit from a common SOM proxy base class 212, which in the preferred embodiment is the Object_SOMProxy class. The SOM proxy base class overrides the " SOMProxy( )" method and causes it to return a pointer to the SOM proxy base class, intstead of a null pointer. This satisfies Case B of FIG. 4.

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L10: Entry 16 of 22 File: USPT Nov 30, 1999

DOCUMENT-IDENTIFIER: US 5995753 A
TITLE: System and method of constructing dynamic objects for an application program

Application Filing Date (1):
19971112

Detailed Description Text (63):
FsmInstance 196 inherits the message interface behavior from FsmEntity 174 and is the base class for state-oriented objects which an application may define, such as call block objects in a telecommunications call processing application. The fundamental object in a telephony application is the call block. It contains the current state of the call for a specific trunk circuit and provides storage for accumulated incoming digits. FsmInstance object class 196 does not contain any application specific behavior or attributes, but instead defines the common behavior for an object which has a state. Application specific objects such as call blocks may be derived from FsmInstance class 196. FsmInstance 196 has an attribute, .sub.-- state, which defines the current state of the object instance, and another attribute, .sub.-- stateMachine, which contains the address of the StateMachine which is used to translate StateIDs into State object instances. When its member function processEvent is called, the received Message is passed to the state through its processEvent member function. This member function returns a pointer to StateID if a state change is required. FsmInstance then issues a request to StateMachine instance to translate the StateID into the address of the state object, calls first the current state's exit member function and then the next state's enter function. FsmInstance further includes a setStateID member function that is a public function used to set the initial StateID, which must be called by FsmInstanceFactory object before the init function is called. Another public function, setStateDictionaryID, sets the objectID of the state dictionary, which must be called before init is called.

Detailed Description Text (94):
FsmDynamicArray class 214 inherits from FsmArray 210 and changes its behavior to support dynamic assignment of FsmEntity to the array. FsmDynamicArray 214 overrides the init method of FsmArray where it adds the index of every entry in the array to the idle queue. Instances of FsmDynamicArray 214 act only as the repository for the array elements. Allocation of idle array entities is performed by instances of FsmDynamicArrayAllocator 194 (FIG. 10).

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L10: Entry 21 of 22 File: USPT Jul 28, 1998

DOCUMENT-IDENTIFIER: US 5787425 A
TITLE: Object-oriented data mining framework mechanism

Application Filing Date (1):
19961001

Detailed Description Text (33):
The common interface of a pure virtual operation definition must be honored by all subclasses such that requesting objects (called client objects) can use subclass member objects (called server objects) without needing to know the particular subclass of the server object. For example, whenever the object defined by the zoo administrator class needs a particular action performed, it interacts with a zoo keeper object. Because the interface to these objects was defined in abstract, base class zoo keeper and preserved in the subclass definitions for the check.sub.--animals() operation, the zoo administrator object need not have special knowledge about the subclasses of any of the server objects. This has the effect of decoupling the need for the action (i.e., on the part of the zoo administrator object) from the way in which the action is carried out (i.e., by one of the objects of the zoo keepers subclasses). Designs (like the ZAP design) that take advantage of the characteristics of abstract classes are said to be polymorphic.

Detailed Description Text (77):
The MiningObject process() method is a virtual method that will be overridden by the process methods specified by the subclasses of the MiningObject class.

Detailed Description Text (80):
The String class is a subclass of ObjectAttribute where the attribute value is a character string. All of the methods in the ObjectAttribute class are virtual methods that will be overridden by the specific methods in the subclasses. The Discrete class is a subclass of ObjectAttribute where the attribute value is an integer value. The Continuous class is a subclass of ObjectAttribute where the attribute value is a floating point value. The String, Discrete, and Continuous classes all provide getValue(), operator=(), operator==(), and setValue() methods. The getValue() and setValue() methods provide an access mechanism for the values of the suing data elements of the object. The operator=() method does an assignment of elementary data. The operator==() method is an equality operator of the elementary data in the member.

Detailed Description Text (124):
The function to be provided in our sample application is that of classification. Classification involves the discovery of the underlying relationships between a set of independent input parameters, and a single dependent output or class variable. The sample data mining application is implemented using the preferred embodiment by subclassing the DataMiningAgent class, created a new ClassificationAgent class. The aDMAgent object in FIG. 19 is an instance of the ClassificationAgent class. The following methods are implemented to override the initialize() and mine() methods in the DataMiningAgent parent class. In addition, a customized control script is written to manage the training and testing of the neural network classifier.

Previous Doc Next Doc Go to Doc#
First Hit

Generate Collection

L10: Entry 2 of 22 File: PGPB Mar 10, 2005

DOCUMENT-IDENTIFIER: US 20050055398 A1
TITLE: Protocol agnostic request response pattern

Detail Description Paragraph:
[0050] Thus, turning to FIG. 8, a block diagram illustrates a class factory 800 with a registry 810 of protocol object creators. The registry 810 of protocol object creators can include identifiers that can be employed to resolve a URI. Such identifiers may be associated with parameters that can be input to an application (e.g., 510, FIG. 5). The registry 810 can be employed to associate identifiers with protocol object creators from protocol objects that have implemented an interface 850 and that in so doing have overridden the one method in the interface 850, create 860. By way of illustration, the creator method 820 may be an implementation of the interface 850 create method 860, with the creator method 820 defined by a protocol object class associated with facilitating HTTP communications. By way of further illustration, the creator method 830 may be an implementation of the interface 850 create method 860, with the creator method 830 being defined by a protocol object class associated with facilitating FTP communications.

CLAIMS:

30. A system that facilitates computer program communication over one of a plurality of protocols, comprising: an input component that receives a communication request from an application; an constructor component that ensures that a requested communication protocol is registered, and that generates at least one protocol object based upon a registered protocol; and a communication component that returns the at least one generated protocol object to the application and communicates with the application through the at least one protocol object via a base class Application Programming Interface (API).

Previous Doc Next Doc Go to Doc#

Previous Doc Next Doc Go to Doc#
First Hit Fwd Refs

Generate Collection

L11: Entry 13 of 15 File: USPT Mar 8, 2005

DOCUMENT-IDENTIFIER: US 6865607 B1
TITLE: Pluggable channels

Brief Summary Text (5):
In distributed object systems, a user application typically has a local representative or proxy to a remote object, where the remote object is often referred to as the server and/or server object. The distributed object system infrastructure typically intercepts method calls made on the proxy, and, in collaboration with infrastructure code delivers the call and parameters associated with the call from the proxy to the server. Similarly, results of the invocation of the call on the server are propagated by the infrastructure from the server back to the proxy, so that to the user it appears that the call executed locally.

Previous Doc Next Doc Go to Doc#

L10: Entry 20 of 22 File: USPT Nov 24, 1998

DOCUMENT-IDENTIFIER: US 5842220 A
TITLE: Methods and apparatus for exposing members of an object class through class signature interfaces

Application Filing Date (1):
19970502

Detailed Description Text (3):
In general, the class signature interface of the present invention provides knowledge of a class and its members. As discussed above, COM interfaces do not expose object class information (e.g., information on the class level). The class signature interface functions to expose class level information. Although the present invention is described for use with the Component Object Model (COM), any object model that requires the use of interfaces or abstract base classes to utilize objects has application for use with the class signature interfaces of the present invention.

Detailed Description Text (14):
In object oriented software development, to facilitate code reuse, a programmer may utilize inheritance capabilities of object oriented programming languages. For example, a programmer may desire some functionality provided through an object defined by a base class. To use the functionality, the programmer, through an inheritance technique, creates a new class, such as derived class, that inherits from the base class. Through inheritance, the derived class has all of the features of the base class. With the derived class, the programmer may either override methods and/or add additional methods and attributes to the derived class.

Generate Collection

L12: Entry 2 of 2 File: PGPB Mar 3, 2005

DOCUMENT-IDENTIFIER: US 20050050548 A1
TITLE: Application internationalization using dynamic proxies

Abstract Paragraph:
An application that was not internationalized when coded may be internationalized through the addition of interception and localization logic and tables without modification of the original application logic. The interception logic may be configured to intercept calls to an application component and invoke localization logic in response to an intercepted call to the application component. The interception logic may use dynamic proxies to intercept method calls from a client component to an application component both before and after the execution of the method. The interception logic may use JAVA reflection to determine whether input parameters or return values associated with the method call are localizable. The application component logic may operate on data stored in a primary database table in which the data is represented in the system default locale.

Summary of Invention Paragraph:
[0009] An application that was not internationalized when coded may be internationalized through the addition of interception and localization logic and tables without modification of the original application logic. The interception logic may be configured to intercept calls to an application component and invoke localization logic in response to an intercepted call to the application component. The interception logic may use dynamic proxies to intercept method calls from a client component to an application component both before and after the execution of the method. The interception logic may use Java reflection to determine whether input parameters or return values associated with the method call are localizable. The application component logic may operate on data stored in a primary database table in which the data is represented in the system default locale. The primary database table may be updated, modified, and maintained by the application component logic using JDBC.

CLAIMS:

7. The system as recited in claim 1, wherein the interception logic includes one or more dynamic proxies that are configured to intercept application component method calls before the execution of the method.

22. The computer accessible medium as recited in claim 21, wherein the program instructions are further executable to call dynamic proxies to intercept the method call to the application component before and after execution of the method.

Previous Doc Next Doc Go to Doc#
First Hit

Generate Collection

L11: Entry 7 of 15 File: PGPB Nov 11, 2004

DOCUMENT-IDENTIFIER: US 20040226001 A1
TITLE: Object-based software management

Detail Description Paragraph:
[0148] The client computer 302 then issues a method call on the interface pointer provided to it by the object creation service (box 908). The proxy object 310 intercepts the method call, generates a notification of the method call to the monitoring system process 320, and forwards the method call to the monitored application object 312 (box 910).

Previous Doc Next Doc Go to Doc#

L11: Entry 12 of 15 File: USPT Aug 9, 2005

DOCUMENT-IDENTIFIER: US 6928488 B1
TITLE: Architecture and method for serialization and deserialization of objects

Brief Summary Text (7):
Conventional distributed object systems may have built-in support for the operations involved in parameter marshalling, which is the packaging of the parameters (call and return) of method calls made on remote objects. Typically, such built-in parameter marshalling is not customizable. In distributed object systems, a user application typically has a local representative or proxy to a remote object, where the remote object is often referred to as the server and/or server object. The distributed object system infrastructure typically intercepts method calls made on the proxy, and, in collaboration with infrastructure code delivers the call and parameters associated with the call from the proxy to the server. Similarly, results of the invocation of the call on the server are propagated by the infrastructure from the server back to the proxy, so that to the user it appears that the call executed locally. Thus, processing involved in remoting a call made on a proxy includes serializing parameters (which may reference objects holding state on the client) associated with the method call.

L11: Entry 14 of 15 File: USPT Jun 8, 2004

DOCUMENT-IDENTIFIER: US 6748555 B1
TITLE: Object-based software management

Detailed Description Text (99):
The client computer 302 then issues a method call on the interface pointer provided to it by the object creation service (box 908). The proxy object 310 intercepts the method call, generates a notification of the method call to the monitoring system process 320, and forwards the method call to the monitored application object 312 (box 910).

CLAIMS:

37. A computer-implemented method of managing a set of object-based programs by collecting operational management information to monitor performance of the set of object-based programs, each of the programs in the set comprising a plurality of monitored software objects, the method comprising: in a runtime environment comprising an object request service and accommodating execution of objects, receiving a runtime request for a reference to an interface of a monitored software object belonging to a program out of the set of object-based programs, the interface of the monitored software object having at least one method for performing a task requested of the monitored software object; responsive to the runtime request, providing, in place of a reference to an interface of the monitored software object, a reference to an interface of a proxy object associated with the monitored software object, the interface of the proxy object accommodating a call to the at least one method for performing a task requested of the monitored software object and operative to forward a method call to the at least one method to the monitored software object; at runtime, intercepting method calls on the monitored software object at the proxy object to direct a first notification indicative of the method call to a software manager and forward the method call to the monitored software object; analyzing the first notification and a second notification generated by another proxy object to generate operational management information comprising at least one metric indicative of program performance; and monitoring the metric to generate an alert when the metric falls outside a threshold.

53. A computer-readable medium having computer-executable instructions for performing a method of managing a set of object-based programs by collecting operational management information to monitor performance of the set of object-based programs, each of the programs in the set comprising a plurality of monitored software objects, the method comprising: in a runtime environment comprising an object request service and accommodating execution of objects, receiving a runtime request for a reference to an interface of a monitored software object belonging to a program out of the set of object-based programs, the interface of the monitored software object having at least one method for performing a task requested of the monitored software object; responsive to the runtime request, providing, in place of a reference to an interface of the monitored software object, a reference to an interface of a proxy object associated with the monitored software object, the interface of the proxy object accommodating a call to the at least one method for performing a task requested of the monitored software object and operative to

forward a method call to the at least one method to the monitored software object; at runtime, intercepting method calls on the monitored software object at the proxy to direct a first notification indicative of the method call to a software manager and forward the method call to the monitored software object; analyzing the first notification and a second notification generated by another proxy object to generate operational management information comprising at least one metric indicative of program performance; and monitoring the metric to generate an alert when the metric falls outside a threshold.

Previous Doc Next Doc Go to Doc#